Programación avanzada en Python

Lección 10: Servicios de red y aplicaciones web

eip ESCUELA INTERNACIONAL POSGRADOS

ÍNDICE

Servicios de red y aplicaciones web

PRESENTACIÓN Y OBJETIVOS

En este último capítulo veremos una pequeña introducción a los servicios de red donde hablaremos de los sockets y como trabajar con ellos.

Por otro lado, también veremos cómo crear un sitio web dinámico con Python

Objetivos

- En esta lección aprenderás a:
- Trabajar con socket
- Crear un entorno virtual
- Crear una aplicación web

1. PROGRAMACIÓN EN RED

Python realiza servicios de red mediante dos niveles de acceso: Nivel bajo y nivel superior

Nivel bajo:

Puede acceder al soporte básico de sockets en el sistema operativo subyacente que implementa clientes y servidores para protocolos orientados a la conexión y sin conexión.

Nivel superior:

Puede acceder a través de librerías Python a protocolos de red específicos de nivel de aplicación como FTP, HTTP, etc.

Sockets

Los puntos finales de un canal de comunicación bidireccional son sockets. Los sockets pueden comunicar procesos entre sí utilizando la misma o distinta máquina. Estos pueden ser implementados usando una variedad de tipos de canales, incluyendo sockets de dominio Unix, TCP y UDP.

Módulo socket

El módulo socket se utiliza para implementar los sockets a través de diferentes canales. La función socket () es utiliza para crear un socket.

Sintaxis:

socket. socket()

Servidor simple

Creamos un objeto socket que utilizaremos para llamar a las otras funciones y realizar la configuración del servidor. Para especificar un puerto para el servicio en el host dado, podemos utilizar la función bind() con dos parámetros nombre de host y puerto. Utilizamos la función accepts para esperar hasta que un cliente se conecte al puerto que se especificó anteriormente. Cuando llega un cliente, retorna un objeto connection que representa la conexión con ese cliente.

Así que un programa de servidor tiene los siguientes puntos:

- Importar el módulo socket
- Crear un objeto socket mediante socket ()
- Obtener el nombre de la máquina local mediante gethostname ()
- Reservar un puerto para su servicio.
- Enlaza con el puerto usando bind ()
- Luego espera la conexión del cliente
- Establezca la conexión con el cliente llegado.
- Por último cerrar la conexión

Programa Python para el servidor:

```
# Import socket module
import socket

# Create a socket object
s = socket.socket()
# Get local machine name
host = socket.gethostname()
# Reserve a port for your service
port = 12345
```

```
# Bind to the port
s.bind((host, port))
# Now wait for client connection
s.listen(5)

while True:
   # Establish connection with client
   c, addr = s.accept()
   print('Got connection from', addr)
   c.send('Thank you for connecting')
   # Close the connection
   c.close()
```

Cliente simple

El programa cliente abre una conexión al puerto 12345 y al host dados. Podemos usar el módulo socket para crear un socket de cliente y usar la función connect () con dos parámetros hostname y port. Abriremos una conexión tcp al nombre del host en el puerto indicado y leeremos cualquier dato disponible del socket.

Una vez que el socket se abre, podemos leer de él. Tendremos que cerrar el socket una vez finalizado. El cliente debe tener los siguientes pasos:

- Importar el módulo socket
- Crear un objeto socket
- Obtener el nombre de la máquina local
- Reservar un puerto para el servicio
- Abre una conexión tcp al nombre de la máquina en el puerto
- Recibe los mensajes tcp
- Cierra el socket al final

Programa Python para el cliente:

```
# Import socket module
import socket

 # Create a socket object
s = socket.socket()
# Get local machine name
host = socket.gethostname()
 # Reserve a port for your service.
port = 12345

# open a tcp conncetion
s.connect((host, port))
# receive tcp messages
print(s.recv(1024))
# Close the socket when done
s.close()
```

Guardamos los dos archivos y ejecutamos uno por uno:

```
# Following would start a server in background.
$ python server.py &

# Once server is started run client as follows:
$ python client.py
```

Resultado:

```
Got connection from ('127.0.0.1', 48437)
Thank you for connecting
```

El proceso se puede entender con el siguiente diagrama:

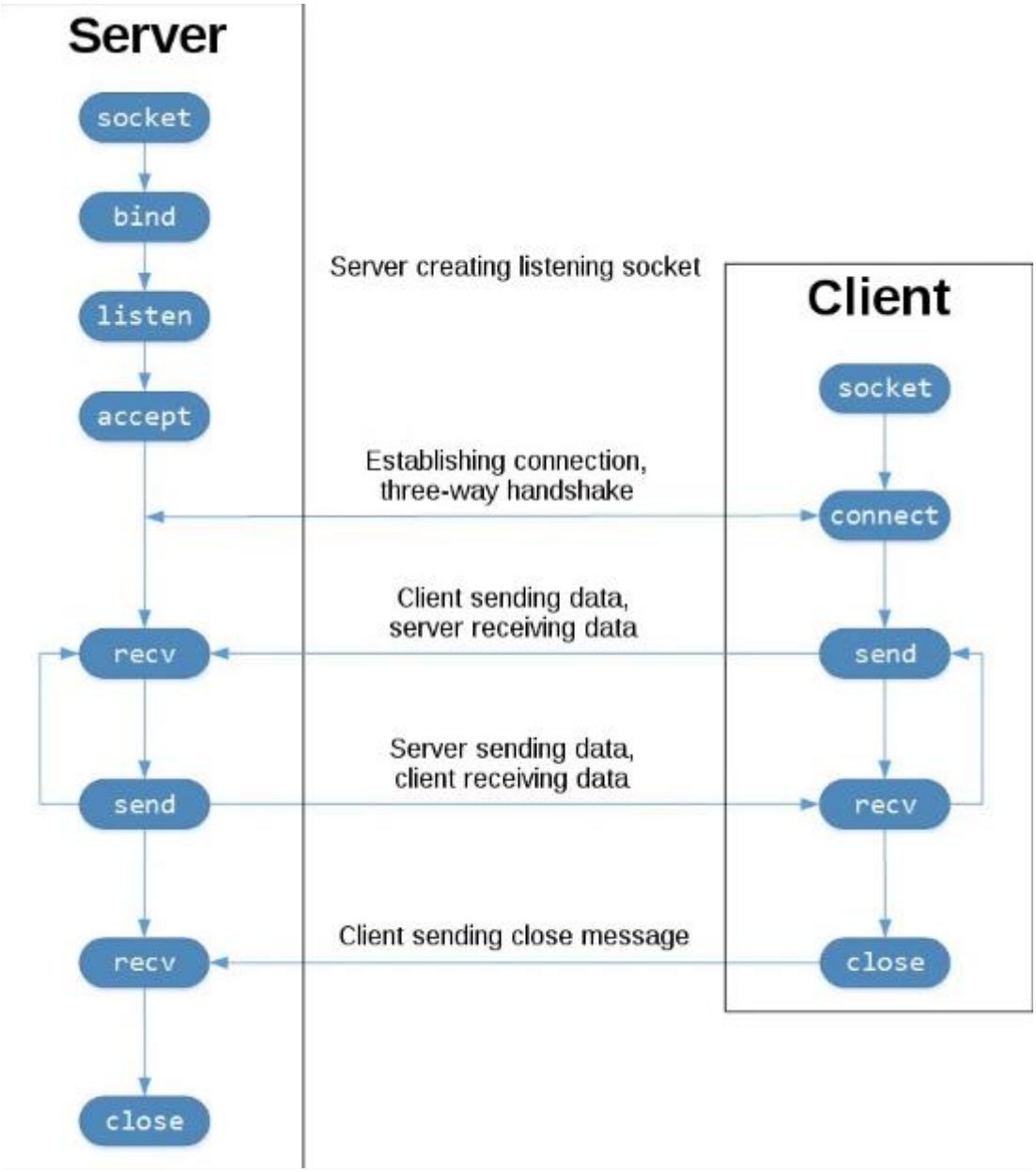

Figura 2.1 Diagrama

2. APLICACIONES WEB

Ahora que somos capaces de construir algunos programas en Python podemos ponerlos a disposición del mundo. Para ello se puede hacer mediante las aplicaciones web.

Hay 3 maneras de distribuir el código:

- Biblioteca Python
- Programa independiente
- Aplicaciones web

Una aplicación web es una de las mejores y tiene la ventaja de ser independiente de la plataforma. Por lo tanto, puede ser ejecutada por cualquier persona que utilice Internet. Hay diferentes tipos de aplicaciones web:

> **Aplicaciones web estáticas:**
>
> En este tipo los sitios web tienen contenidos fijos y su contenido no cambia cuando se interactúa con él. No se consideran aplicaciones porque no son dinámicas.

> **Aplicaciones web dinámicas:**
>
> Las aplicaciones web dinámicas son consideradas como verdaderas porque actúan dinámicamente y cambian su contenido al interactuar con ella. La aplicación de correo web es uno de los ejemplos que permiten al usuario interactuar con ella de muchas maneras y recibimos los correos a medida que van llegando.

Construir una aplicación web básica

Vamos a ver cómo desarrollar una simple aplicación web que soporta el entorno de Python, con un framework llamado flask

Configurar el proyecto

Creamos una carpeta de proyecto y le asignamos un nombre que sea descriptivo de nuestro proyecto. Por ejemplo llamemos a la carpeta hello-app. Necesitaremos dos archivos dentro de esta carpeta:

1. main.py contiene tu código Python envuelto en una implementación mínima del framework web Flask.
2. requirements.txt lista todas las dependencias que tu código necesita para funcionar correctamente.

main.py

main.py es el archivo que Flask utiliza para entregar el contenido. En la parte superior del archivo, importamos la clase Flask en la línea 1, y luego creamos una instancia de una aplicación Flask en la línea 3:

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def index():
    return "Congratulations, it's a web app!"
```

Después de crear la aplicación Flask, escribimos un decorador de Python en la línea 5 llamado @app.route que Flask utiliza para conectar los 'endpoint' de la URL con el código contenido en las funciones. El argumento de @app.route define el componente de la ruta de la URL, que en este caso, es la ruta raíz ("/").

El código de las líneas 6 y 7 definen la función index(), que está envuelto por el decorador. Esta función define lo que debe ejecutarse si el 'endpoint' de la URL definida es solicitado por un usuario. Su valor de retorno determina lo que el usuario verá cuando cargue la página

requisitos.txt

En este fichero escribriremos todos los paquetes que nuestra aplicación necesita para funcionar. Flask es la única dependencia de este proyecto, eso es todo lo que necesitamos especificar:

```
Flask==1.1.2
```

Probando nuestra web

Flask tiene incorporado un servidor web de desarrollo. Podemos utilizar este servidor de desarrollo para comprobar que nuestro código funciona como se espera. Para poder ejecutar el servidor de desarrollo de Flask localmente, necesitas completar dos pasos.

1. Configurar un entorno virtual.
2. Instalar el paquete flask.

Para configurar un entorno virtual de Python 3:

```
$ python3 -m venv venv
```

Necesitas activar el entorno virtual mediante el script de activación:

```
$ source venv/bin/activate
```

Una vez configurado y activado nuestro entorno virtual, instalamos las dependencias:

```
$ python3 -m pip install -r requirements.txt
```

Este comando obtiene todos los paquetes listados en requirements.txt y los instala en el entorno virtual. Esperamos a que se complete la instalación, finalmente, abrimos main.py y añadimos las siguientes dos líneas de código al final del archivo:

```
if __name__ == "__main__":
    app.run(host="127.0.0.1", port=8080, debug=True)
```

Por último ejecutamos el script de Python que inicia la aplicación Flask escribiendo el siguiente comando:

```
$ python3 main.py
```

```
* Serving Flask app "main" (lazy loading)
* Environment: production
  WARNING: This is a development server.
  Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on http://127.0.0.1:8080/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 315-059-987
```

3. PUNTOS CLAVE

- Un **socket** es un tipo de punto final que funciona para establecer un enlace de red bidireccional entre un servidor y un cliente.
- **Virtualenv** es una herramienta para crear entornos Python aislados.
- **Flask** es un marco de aplicación web ligero . Está diseñado para que la puesta en marcha sea rápida y sencilla, con la capacidad de escalar a aplicaciones complejas.